1 Apprentissage par renforcement (reinforcement learning)

1.1 Généralités

Le reinforcement learning est une partie importante de la théorie de l'apprentissage, et qui a été utilisée par la société Google Deepmind [3] pour leur IA de Go. Cet technique met en jeu un agent qui évolue dans un environnement (ici non stochastique) en prenant des décisions. Ces décisions le font passer d'un état à un autre en recevant une récompense (reward). Il s'agit alors de trouver une stratégie qui maximise la récompense cumulée.

On modélise mathématiquement ceci par un processus de décisions markovien.

Soit S l'ensemble des états définissant l'environnement et A un ensemble d'actions.

On suppose l'environnement déterministe, c'est-à-dire que l'on se donne une fonction de transistion :

$$\begin{array}{rccc} T: & S \times A & \longrightarrow & S \\ & (s,a) & \longmapsto & s' = T(s,a) \end{array}$$

Pour $(s,a) \in S \times A$, $T(s,a)$ est l'état dans lequel se retrouve l'agent en partant de s et en agissant avec l'action a. Il reçoit également une récompense que l'on note $R(a,s)$. On définit donc la fonction récompense (supposée également déterministe) :

$$\begin{array}{rccc} R: & S \times A & \longrightarrow & \mathbb{R} \\ & (s,a) & \longmapsto & r = R(s,a) \end{array}$$

Une fonction $\pi : S \longrightarrow A$ qui à un état associe l'action à prendre dans cet état définit la politique π.

Considérons maintenant un état initial $s_0 \in S$, obtenue aléatoirement par une distributions ν sur S. En suivant une politique π à partir de s_0, on évolue dans l'environnement, et on obtient une suite d'états $(s_t)_{t\in\mathbb{N}}$ et une suite de récompense $(r_t)_{t\in\mathbb{N}}$ définis par les relations :

$$s_{t+1} = T(s_t, \pi(s_t)) \text{ et } r_t = R(s_t, \pi(s_t))$$

Pour $\gamma > 0$, on définit la récompense cumulée par :

$$R^{\pi}_{s_0} = \sum_{t=0}^{\infty} \gamma^t r_t$$

(il y a d'autres définitions possibles comme $\sum_{t=0}^{h} r_t$, avec l'horizon h fixé, ou encore $\overline{lim}_{h\to\infty} \frac{1}{h} \sum_{t=0}^{h} r_t$.)

L'objectif du reinforcement learning est alors de calculer une politique π qui maximise $L(\pi) = \mathbb{E}_\nu(R^\pi_{s_0})$ où s_0 a pour distribution ν. D'après [1], cette politique optimale existe toujours.

Pour cela, on définit deux quantitées importantes :

- le gain obtenu par l'agent lorsqu'il suit π depuis s :
$$\begin{array}{rccc} V^\pi : & S & \longrightarrow & \mathbb{R} \\ & s & \longmapsto & R^\pi(s, \pi(s)) \end{array}$$
- le gain obtenu par l'agent lorsqu'il agit selon une action a à partir de s_0 puis suit la politique π :
$$\begin{array}{rccc} Q^\pi : & S \times A & \longrightarrow & \mathbb{R} \\ & (s, a) & \longmapsto & R^\pi(s, a) + \gamma V^\pi(T(s, a)) \end{array}$$

On a alors les relations

$$V^\pi(s) = Q^\pi(s, \pi(s)) \text{ et } Q^\pi(s, a) = R^\pi(s, a) + \gamma V^\pi(T(s, a))$$

Notons π^* une politique optimale et V^*, Q^* les fonctions associées. Alors $V^*(s) = Q^*(s, \pi^*(s))$. Comme $a \longleftarrow Q^*(s, a)$ est maximale en $\pi^*(s)$ car π^* est maximale et le processus est markovien (c'est-à-dire que les états passés n'ont pas d'influence sur le futur), on a :

$$\pi^*(s) = \underset{a\in A}{\operatorname{argmax}}\, Q^*(s, a)$$

Il suffit donc de déterminer Q^* pour déterminer π^*.

Ceci est rendu possible par l'équation de Bellman, que nous allons établir. Pour π une politique, en notant $s' = T(s,\ \pi(s))$ et $r = R(s, a)$, on a :

$$\begin{aligned} Q^\pi(s, a) &= r + \gamma V^\pi(s') \\ &= r + \gamma Q^\pi(s', \pi(s')) \end{aligned}$$

Pour $\pi = \pi^*$:

$$\begin{aligned} Q^*(s, a) &= r + \gamma Q^*(s', \pi^*(s')) \\ &= r + \gamma \max_{a'\in A} Q^*(s', a') \end{aligned}$$

On a ainsi établit l'équation de Bellman, qui ne fait intervenir que la fonction Q^* :

$$Q^*(s,a) = r + \gamma \max_{a' \in A} Q^*(s',a')$$

Cette équation nous permet de calculer Q^* et donc π^*, par programmation dynamique.

Partons d'une politique initiale π_0 quelconque. On construit une suite de politque (π_n) et la suite (Q_n) associée par les relations de récurrences :
- si $((s_t^n)_t, (a_t^n)_t, (r_t^n)_t)$ est un épisode d'un agent qui a suivi la politique π_n alors on pose :

$$Q^{n+1}(s_t^n, a_t^n) = r_t^n + \gamma \max_{a' \in A} Q^n(s_{t+1}^n, a')$$

$$Q^{n+1}(s,a) = Q^n(s,a) \text{ pour } (s,a) \notin \{(s_t^n, a_t^n)_t, t \geq 0\}$$

et

$$\pi^{n+1}(s) = \operatorname*{argmax}_{a \in A} Q^{n+1}(s,a)$$

On démontre dans [9] que Q_n converge vers Q^* et que π_n converge vers π^*.

Cependant, cette méthode n'est pas envisageable si S ou A sont des ensembles infinis, ou même de cardinal grand. Dans [7], l'équipe de DeepMind a proprosé une nouvelle méthode utilisant des réseaux de neuronnes : les DQN (Deep Q-network).

Il s'agit d'approximer Q^* par un réseau de neuronnes. On définit pour un réseau de neuronnes de poids θ, la fonction associée $Q(\cdot, \cdot, \theta)$.

L'objectif est d'approcher Q^* par les $Q(\cdot, \cdot, \theta)$. Pour cela, on définit une suite de poids (θ_n) tel que θ_{n+1} minimise la fonction de coût :

$$L_{n+1}(\theta_{n+1}) = \mathbb{E}_{(s,a)\sim\rho}(y_{n+1} - Q(s,a,\theta_n)^2)$$

avec ρ une distribution sur $S \times A$ et

$$y_{n+1} = R(s,a) + \gamma \max_{a' \ \in A} Q(s',a',\theta_n) \text{ ,où } s' = \pi^n(s)$$

On montre alors :

$$\nabla_{\theta_{n+1}} L_{n+1}(\theta_{n+1}) = \mathbb{E}_{(s,a)\sim\rho}((r + \gamma \max_{a' \in A} Q(s',a',\theta_n))\nabla_{\theta_n} Q(s,a,\theta_n))$$

On peut donc utiliser cette formule pour effectuer une méthode d'optimisation (descente de gradient par exemple), et ainsi calculer θ_{n+1}.

1.2 Application aux enchères du bridge

Dans [6] et [11], une application du reinforcement learning est proposé pour la génération de séquences d'enchères au bridge. Nous allons présenter et implémenter une version simplifiée de ce qu'il a été proposé.

1.2.1 Définition du problème

Le problème d'enchères peut être défnit de la manière suivante : quatre joueurs répartis en deux équipes (Nord-Sud ou NS) et (Ouest-Est ou WE), on chaque joueur annonce un contrat, supérieur au précedent, jusqu'à ce que trois joeurs concécutifs passent. Le contrat à remplir est alors le dernier contrat annoncé. Pour simplifier, on suppose des enchères sans compétitions, c'est-à-dire que l'on se place du point de vue de NS, et que WE passe constamment.

Mathématiquement, on peut modéliser cela de la manière suivante. On note $\mathbb{B} = \{1\clubsuit < 1\diamondsuit < 1\heartsuit < 1\spadesuit < \cdots < 7\spadesuit < 7NT \ < PASS$ où NT signifie sans atouts, l'ensemble des contrats (ordonnés dans cet ordre). Une séquence d'enchères b valide est alors une suite finie croissante de $\mathbb{B}$ dont le dernier élément est PASS (en effet comme WE passe, les enchères s'arrêtent dès que N ou S passe). Le contrat à remplir sera alors l'avant dernier élément de b. Par exemple $b = (2\heartsuit, 4\diamondsuit, 6\heartsuit, PASS)$ est une séquence d'enchères valide et N devra remplir le contrat $6\heartsuit$.

Notons x_N (resp. x_S) les cartes de N (resp. de S). On définit alors pour $t \in \mathbb{N}$, $x^t = x_N$ si t pair et $x^t = x_S$ si t impaire.

Le problème est alors de trouver une stratégie $G : (x_N, x_S) \longleftarrow \{b \in B^{\mathbb{N}} \text{ valide}\}$ qui donne le meilleur contrat possible.

1.2.2 Modélisation en processus markovien

On veut modéliser le problème par un processus markovien : il s'agit de définir S, A T et R.

À la t-ième étape de l'enchère, on définit l'état comme :

$$s = (x^t, b^0, \ldots, b^{t-1})$$

On pose alors $A = \mathbb{B}$ et pour $(s, a) \in S \times A$:

$$T(s, a) = (x^{t+1}, b^0, \ldots, b^{t-1}, a)$$

si $(b^0, \ldots, a)$ est une séquence d'enchères valide. Si cette séquence n'est pas valide, alors on reste dans le même état :

$$T(s, a) = s$$

De plus, si $a = PASS$, l'épisode s'arrête.

Définissons maintenant le système de récompense. Pour cela, nous devons introduire l'algorithme de Double Dummy Analysis (DDA) [5]. Il s'agit d'un algorithme qui étant donnée une répartition de cartes entre NS et WE, calcul le score maximal (en nombre de passes) que chacun que atteindre pour un contrat donné. Par exemple, pour la répartition suivante :

N	♠K983	♡Q62	♢K10832	♣K
E	♠QJ754	♡K987	♢	♣10865
S	♠1062	♡AJ10	♢AJ654	♣92
W	♠A	♡543	♢Q97	♣AQJ743

l'algorithme renvoie les scores suivants :

	NT	♠	♡	♢	♣
N	6	6	6	10	3
S	6	6	6	10	3
E	5	5	7	3	10
W	5	5	7	3	10

On peut maintenant définir la fonction récompense R.

- si $a \neq PASS$, alors $R(s, a) = 1$ si la séquence d'enchères est valide, 0 sinon
- si $a = PASS$, alors $R(s, a) = \frac{100}{1+n_{max}-n} - l$ où n_{max} est le score maximal pour cette répartition de cartes (obtenu par DDA), n le score obtenu pour le contrat déterminé par la séquence d'enchères, et l la longueur de la séquence d'enchères.

Ainsi, si l'action déterminée par la politique mène à une séquence invalide alors la récompense est nulle et l'état reste le même. Si l'action mène à une séquence valide, alors on récompense cela par un gain de 1. Quand l'enchère est terminée (c'est-à-dire que l'action est $PASS$), alors plus le contrat est bon (c'est-à-dire proche du score maximal déterminé par DDA), plus la récompense est grande. On retire la longueur des enchères pour que le score cumulé soit exactement cette récompense.

1.2.3 Implémentation

Nous avons utilisé les implémentations open-source de l'entreprise OpenAI [10], ainsi qu'une implémentation en C++ du DDA :

- [2] permet d'implémenter un processus décisonnel markovien en tant qu'environnement.
- [4] implémente les algorithmes relatifs aux DQN.
- [8] implémente le DDA

1.2.4 Résultats

Références

[1] Frederick J. Beutler and Keith W. ROss. Optimal policies for controlled markov chains with a constraint. *Journal of Mathematical Analysis and Applications*, 1983.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] DeepMind. https ://deepmind.com.

[4] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https ://github.com/openai/baselines, 2017.

[5] Bo Haglund. Double dummy solver. http ://privat.bahnhof.se/wb758135/bridge/index.html, 2004.

[6] Chun-Yen Ho and Hsuan-Tien Lin. Contract bridge bidding by learning. *Proceedings of the Workshop on Computer Poker and Imperfect Information at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[8] Eric Seidel. Bo haglund's double dummy solver. https ://github.com/eseidel/dds, 2013.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[10] OpenAI team. https ://openai.com.

[11] Chih-Kuan Yeh and Hsuan-Tien Lin. Automatic bridge bidding using deep reinforcement learning. *CoRR*, abs/1607.03290, 2016.